THE UNIVERSITY OF
MELBOURNE

Department of
Computer Science and Software Engineering

Software Design Document

for

Managing the analysis of massive DNA sequence data

Version: 3.0
October 30, 2009

This document specifies the architectural design of the system. The design of the system is explained in detail to provide the development team a guideline to follow in order to meet design goals and critical software requirements.

CONTENTS

1 Introduction **1**

1.1 Project Overview 1

1.2 Project Scope 2

1.3 Audience of the Document 3

1.4 Document Preview 3

2 Architectural Design **4**

2.1 General Constraints 4

2.2 Design Goals 4

2.3 Data Description 5

2.4 Architectural Design Model 7

2.5 Alternative Considered 14

3 Detailed Design **15**

3.1 Class Diagram 17

3.2 Sequence Diagrams 19

ONE

Introduction

1.1 Project Overview

1.1.1 The Client

The client of the project team is Jason Li who represents Peter MacCallum Cancer Center. One of the research activities conducted by the biologists of Peter MacCallum Cancer Centre is to regularly compare short fragments of sampled DNA sequences to an existing library of known DNA sequences. This allows biologists to study genetic activities that have occurred in the sampled individual. BLAST is used to carry out the sequence comparison process. Prior to applying BLAST, data is trimmed as desired to remove junk data and also to be manipulated so as to extract the regions of interest that have occurred in the sampled individual. One type of such regions of interest is the unique identifier that was attached to the sequences during the experiment and the other type is the region to be compared to the library of known DNA sequences. A piece of software has been requested to be built in order to simplify their job and to reduce operational costs.

1.1.2 Root Causes

- In order to match DNA sequences, R scripting is used to run command lines that carry out the BLAST process. This means the sequencing process takes several hours to generate results. The software in development will display the first set of results to allow the user to make modifications if desired as well as aiming to reduce the total time taken by breaking samples down and sequencing them in multiple parallel processes.

- Most biologists are also unable to carry out this process of grouping and selecting DNA sequences on their own as they do not have the technical knowledge nor the experience. Therefore, queries are passed to Jason to be processed before being run through BLAST. This causes one person to be responsible for all the queries that are sent by the biologists. The software in development will allow the biologists to sequence data on their own individual computers, as each individual biologist will have his own stand-alone computer (with the software) to work with.

- At the moment, only the client has the expertise and the experience to sequence the data on his own. Many biologists, except for several bioinformaticians on-site, do not have the clients experience and so are unable to assist him in writing scripts for sequencing data. This causes delays in accessing the results, and also in their research.

The software will enable biologists and bioinformaticians to easily adapt to and learn the processes of sequencing data without imposing the responsibility on one single individual.

1.1.3 Users of the System

The main users of the system will be:

- biologists
- statisticians
- clinicians
- bioinformaticians

All users are expected to have prior basic computer knowledge as well as sufficient knowledge in their field to BLAST sequences. Users are also expected to know the following information.

- Names and location of input and reference files
- If an input file has a paired end
- Values to set the quality threshold
- Groups that should be formed
- Unique identification region
- Columns for the output table
- BLAST region in the input and reference files
- Parameter values for BLASTing
- Number of threads
- Location to save files

1.2 Project Scope

The system developed is a GUI, which wraps around a family of algorithms for comparing primary biological sequence information called BLAST. This system will reduce the workload and the time for the bioinformaticians to run DNA mapping queries against existing libraries of DNA sequences. Biologists, in particular are required to study genetic activities that have occurred in the sampled individual. To be able to work efficiently, their queries are usually submitted to any bioinformatician in their lab while they could be studying. With the new system in place, bioinformaticians are free from this task so they can focus on their own work while the biologists can do the querying in their own time without affecting their study and bothering their colleagues.

This system provides a GUI to pre-process data such as trimming input data, grouping required short reads and so on, before allowing the user to execute BLAST. The old system requires bioinformaticians to manually use R-scripting and command-line processing was too overwhelming for users who are not so familiar with command-line systems. The user will now be able to easily work their way through the system in development. Hence the end system will be user friendly for the bioinformaticians as well as the biologists.

1.3 Audience of the Document

This document is targeted towards the development team so the developers will have a feasible design of a system to implement. The document is also meant for the maintenance team and the client to refer to in the future so they will have a good understanding of the system for when they are to change any part of the system.

1.4 Document Preview

This document begins by identifying the general constraints the design team has to design around, and also the design goals the team aims to accomplish. A description of the data the system handles is included.

Subsequently, the architectural design is described in detail. Modules and their responsibilities are described, and the way each module communicates with each other is also explained.

The design team had to decide between two options when coming up with the architecture. The alternative that the team decided to go against is described in this section.

TWO

Architectural Design

This section provides an overview and the rationale of the data and architectural design decisions made by the development team. General constraints and design goals have been identified and explained. Stucture and formats of external files and internal file has also been described. Finally, architectural model has been selected and its major components has been described and justified in relation to the project.

2.1 General Constraints

The following constraints are listed in order of importance (highest to lowest):

2.1.1 Platform Compatibility

The client has requested the system to be able to run on variety of platforms. The system must be able to run on at least two different operating systems, mainly Windows XP and and Mac OS X Leopard. This design goal is achieved by programming in Java.

2.1.2 Extensibility

The client has requested for the system to be extensible in the future. The system currently wraps around the existing BLAST program, and it should be simple for any maintenance team in the future to be able to replace the BLAST program with another working program. Furthermore, it should also be easy for the maintenance team in the future to be able to change any part of the program. This design goal will be achieved with the use of well-documented code and the adherence to programming conventions.

2.2 Design Goals

The following design goals are listed in order of importance (highest to lowest):

2.2.1 Performance

A standard BLAST process can take a long time, and an important design goal is to reduce the time taken to BLAST input data. The utilisation of multiple threads will be able to achieve this. This will shorten the amount of time necessary to BLAST a certain amount of data because data will be processed using several BLAST processes running in parallel.

2.2.2 Usability

The user interface will be very simple to learn and to use. The design aims to achieve such usability by providing a very simple interface for the user to interact with. The user will be able to carry out pre-processing tasks very easily using the user interface. The steps leading up to the activation of BLAST will also be straightforward and simple. This design goal can be achieved using short but descriptive on-screen instructions.

2.3 Data Description

Data can be used and/or stored by the system in multiple ways. However, the current architecture of the system handles them in a particular way. This section describes the data that will be used and/or stored by the system. The usage of these data is also briefly identified.

Persistent data required to be stored is a library of short read sequences stored in one or more references files. Reference files (CSV format) are estimated at around 1.4GB (∼1400MB) stored in the user's local computer. Peter MacCallum Cancer Centre expects to receive updates of approximately 70,000 to 10,000 DNA sequences from external institutes every few months.

2.3.1 Schema

Data Type: Database File
Size: ∼1.4GB
Number of Records: 1-5
Data it keeps: DNA Sequences
Why it is required: To be used for comparing against query strings
Parts of the software writes: None
Update Time: When PMCC receives new data
Estimate of Update Frequency: 3-4 months.

2.3.2 File and Data Formats

2.3.2.1 Input Files

Input files are:

- generally 700MB, and may come in pairs.
- in TXT format.
- is colon seperated in columns.
- *a.k.a. query files*

Our system will primarily deal with the 6th and 7th column (short read and quality, respectively).

Our system will assume that:

- there will always be 7 columns.
- the short read will always be the 6th column.
- the quality will always be the 7th column.

2.3.2.2 Reference Files

Reference files are:

- in CSV format
- may be linked over several files (due to Excel 2003's row limit of 65535)
- *a.k.a. database files*

The new system will allow the user to choose columns which will be displayed on the output to help users understand.

The new system will assume that each reference file has a header row.

2.3.2.3 Tag Files

Tag files are generated by the new system and are in TXT format (tab delimited).

The new system will generate tag files based on the count of each short-read's occurence. Each line of the tag file will be printed as such: %d\t\t%s. The filename of the tag file will be *$input_filename.tag*

2.3.2.4 FASTA Files

FASTA files are generated by the new system and are in FASTA format.

A FASTA file generally looks like:
>description
short-read

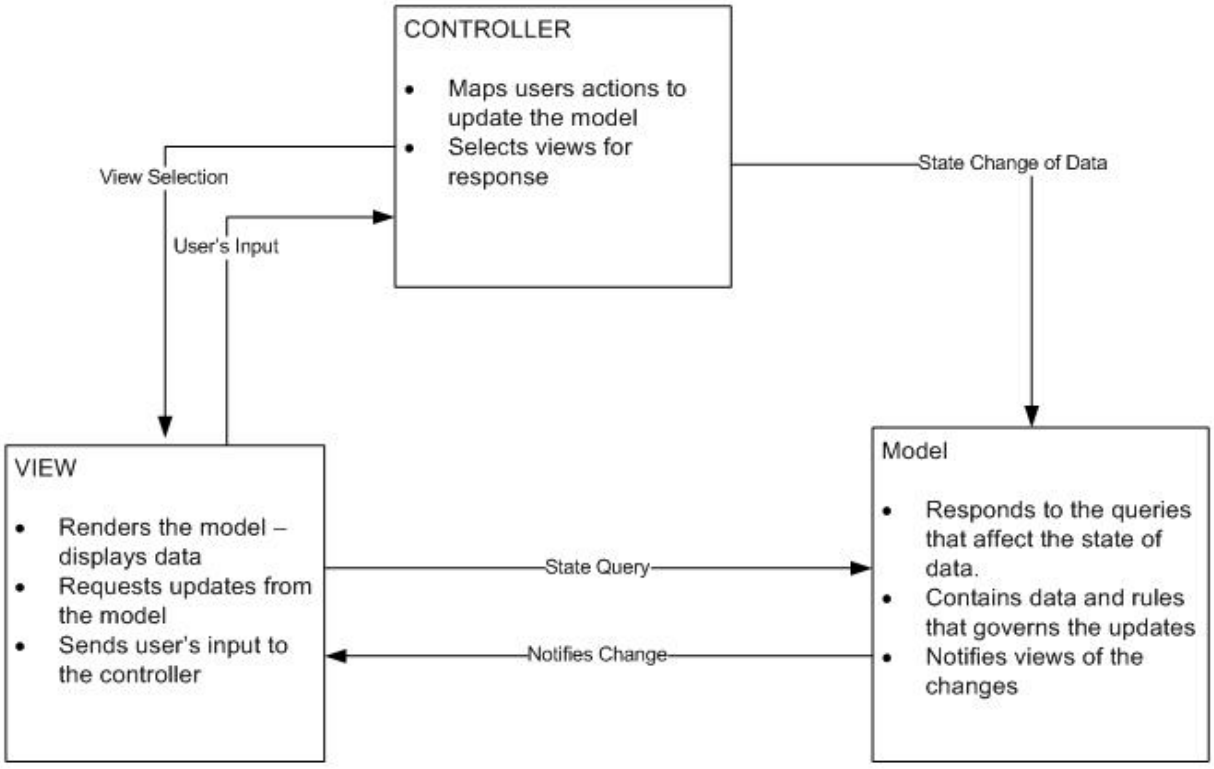
The new system will generate FASTA files which look like
>$ID_$count_$short_read
$short_read

Each FASTA file will be named $ID.fa and will only contain short-reads which have the identification $ID.

2.4 Architectural Design Model

The system employs the model-view-controller (MVC) architecture. The MVC style fits the purpose of handling data manipulation in the back-end of the system and receiving user inputs from the interface while displaying required information and contents to the user.

The Model contains the data and the rules that govern the access to the data and also controls the updating of the data. The View displays the contents of the Model. How data should be presented is specified in the view. If data is changed, the View must update the Model as needed. The controller maps the user's interaction with the system into commands that the Model will execute.



The MVC architecture is suitable for this particular project as the development team is familiar with this type of architecture, since it is very commonly used in software engineering. This architecture separates the user's inputs, actions and the system's display from the actual data and processing that occurs in the background. This ensures that the code is well organised and encapsulated. The encapsulation of the internal modules in each of the MVC component will ensure that the end program is extensible.

The MVC architecture will also induce low coupling and high cohesion into the end

system. This again will ensure the program is extensible as modules are isolated and a change in one module will require very little or even no change to the implementation of another module.

The separation of the presentation logic from the business logic of the system makes it easier for the team to be able to meet the usability design goal. This is because the team will be able to easily change the user interface at a later stage of the implementation stage without the need to make major changes to the business logic. This allows the same information to be presented to the user in multiple ways.

The extensibility of the software is a requirement as requested by the client, who has stated that he may want to work on the software on his own in the future.

2.4.1 Domain Model of MVC Architecture

The domain model is divided into three sections to reflect the modules that will be present in the Model, View and Controller components of the MVC architecture that the design employs. The design however uses a slightly modified version of the original MVC model.

The Model component handles the storage of information. Information is manipulated by the Controller component. The Model contains information like files, BLAST parameters, identifications, groups, etc. One big difference this Model has from a conventional Model in an MVC architecture is that the Model employed in this system carries out processing operations. BLAST is called from the Model, and BLAST's inputs are also separated into threads in the Model.

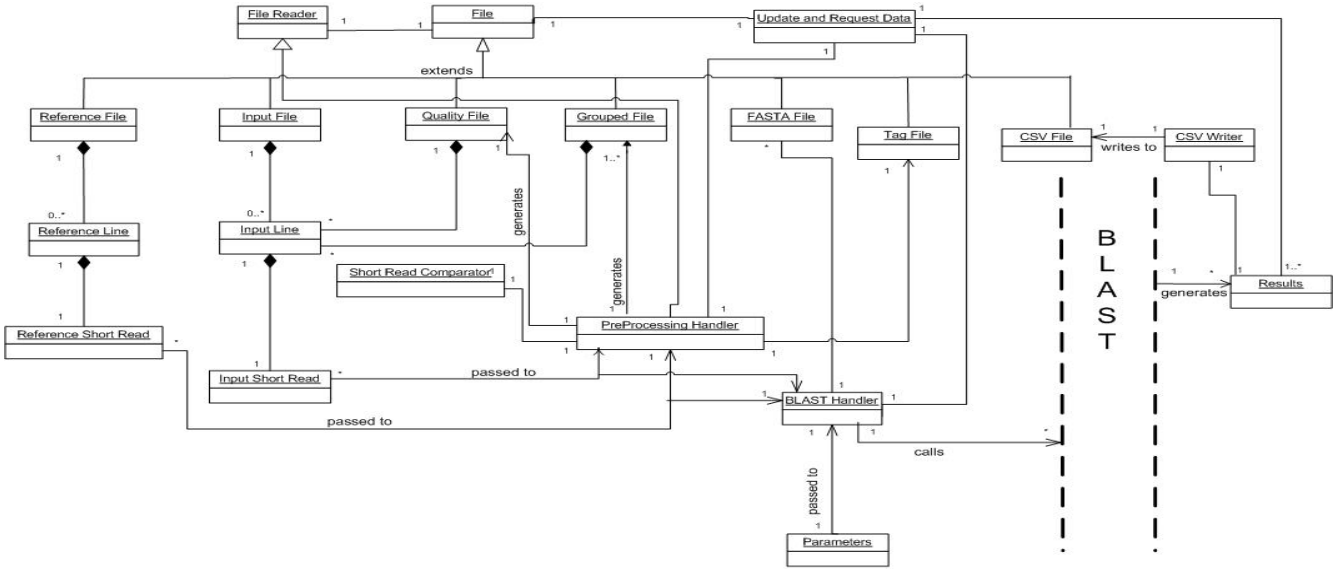
The View component handles the graphical user interface of the system. It contains panels that form the interface, and other components the user interface consists of, e.g. buttons, textboxes, etc.

The Controller component handles the user's interaction with the user interface. It calls specific attributes and methods from the Model when certain user interactions with the View are made. An important thing to remember is that the Controller does not retrieve data from the Model.

2.4.2 Domain Model of the Model component of the architecture

The domain model below describes the inner implementation of the Model component of the architecture. The model component acts as database to the system and interacts with BLAST. The model contains two main classes that executes processing requested by the controller: The Pre-Processing Handler module will handle the processing for the pre-processing flow of the system. It will use reference file modules and input file modules to generate grouped files and quality files. The BLAST Handler module will handles the BLAST processing flow of the system. It will generate FASTA files using BLAST, and it will feed BLAST inputs into BLAST for the actual BLAST processing. Data that are to undergo the actual BLAST processing will be separated into multiple threads that will be fed into multiple BLAST processes. For instance, if the user specifies four threads to be used for the BLAST process, the system will BLAST four threads of data at once, while other data stays in a queue. Once a thread is finished, another

thread of data is sent from the queue to be processed in parallel with three other BLAST processes.



2.4.2.1 Model: Modules and Responsibilities

2.4.2.1.1 Model Interface Module

- Responsibility: To accept requests and queries from the model and the controller to send and update data such as parameters, groups and short reads in input and reference files

2.4.2.1.2 Reference File Module

- Responsibility: To read and store data from the reference file

2.4.2.1.3 Reference Line Module

- Responsibility: To store each row in the reference file

2.4.2.1.4 Input File Module

- Responsibility: To read and store data from the input file

2.4.2.1.5 Input Line Module

- Responsibility: To store each row in the input file

2.4.2.1.6 Short Read Module

- Responsibility: To store the short read in each input line

2.4.2.1.7 PreProcessing Handler Module

- Responsibility: To generate tag files, FASTA files, grouped files and quality files

2.4.2.1.8 Quality File Module

- Responsibility: To store the output from quality filtering

2.4.2.1.9 Grouped File Module

- Responsibility: To store the output from grouping

2.4.2.1.10 FASTA file Module

- Responsibility: To store FASTA files generated by the system so that they can be used as inputs into BLAST

2.4.2.1.11 Tag file Module

- Responsibility: To store a tag file generated by our system, which contains an identification of each Short Read object and the number of times it has appeared in the input files.

2.4.2.1.12 CSV File Module

- Responsibility: To store the results produced by BLAST in a CSV File

2.4.2.1.13 Parameters Module

- Responsibility: To store parameters to be used during the BLAST process

2.4.2.1.14 Results Module

- Responsibility: To Display the results of the BLAST process

2.4.2.1.15 BLAST Handler Module

- Responsibility: To generate FASTA files and acts as an interface between the model and the BLAST program

2.4.2.1.16 CSV Writer Module

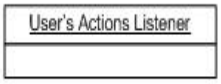
- Responsibility: To write the results into a CSV file

2.4.2.1.17 CSV Reader Module

- Responsibility: To read CSV files

2.4.3 Domain Model of the Controller component of the architecture

This one class will handle the mapping of user's interaction with the system to the system. It will execute instructions in response to user's input and gestures.

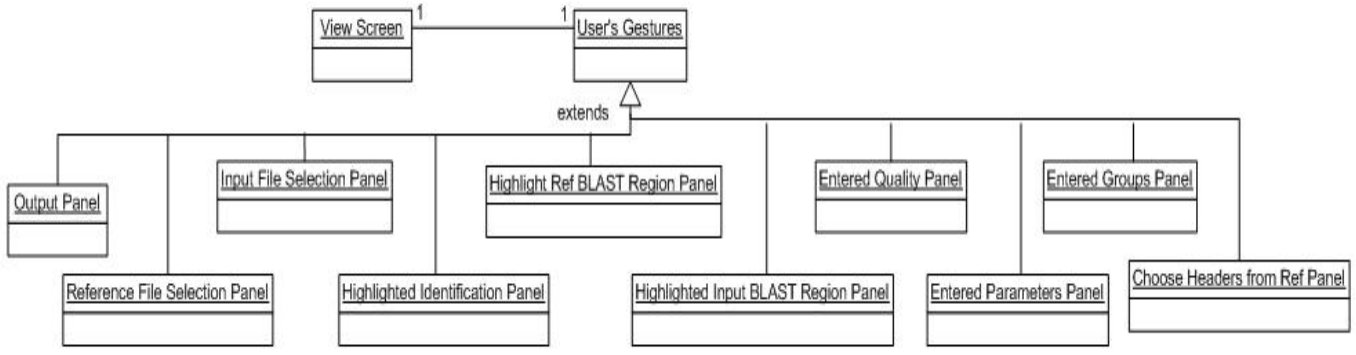


2.4.3.1 Controller: Modules and Responsibilities

2.4.3.1.1 Controller Interface Module

- Responsibility: To maps user's inputs and actions to the Model to update and process data in the model

2.4.4 Domain Model of the View component of the architecture.



2.4.4.1 View: Modules and Responsibilities

2.4.4.1.1 View Screen Module

- Responsibility: To send requests to the Model and receives updated data that is to be viewed

2.4.4.1.2 User's Gestures Module

- Responsibility: To send user's actions to the Controller to be processed

2.4.4.1.3 Output Panel Module

- Responsibility: To display the output to the user and allow the user to save the output

2.4.4.1.4 Reference File Selection Panel Module

- Responsibility: To allow the user to select a reference file

2.4.4.1.5 Input File Selection Panel Module

- Responsibility: To allow the user to select an input file

2.4.4.1.6 Highlighted Identification Panel Module

- Responsibility: To allow the user to view the first 20 lines of the input file and highlight a section of the line to be the identification

2.4.4.1.7 Highlighted Ref BLAST Region Panel Module

- Responsibility: To allow the user to view the first 20 lines of the reference file and highlight the BLAST region

2.4.4.1.8 Highlighted Input BLAST Region Panel Module

- Responsibility: To allow the user to view the first 20 lines of the input file and highlight the BLAST region

2.4.4.1.9 Entered Quality Panel Module

- Responsibility: To allow the user to view and enter quality thresholds

2.4.4.1.10 Entered Parameters Panel Module

- Responsibility: To allow the user to view default parameters and enter values

2.4.4.1.11 Entered Groups Panel Module

- Responsibility: To allow the user to view and enter group strings

2.4.4.1.12 Choose Headers from Ref Panel Module

- Responsibility: To allow the user to view the headers and select particular column headers

2.4.5 Interfaces

2.4.5.1 Model Interface

- 'Model Interface Module' is an interface to the Model that communicates with the Controller and the View components of the architecture. This interface communicates with the Controller through the class 'Controller Interface Module' to receive instructions to update data according to the users actions. This interface also communicates with the View through the class 'View Screen' in order to notify the View component of the updates that have been made to the data which is viewed.

2.4.5.2 Controller Interface

- 'Controller Interface Module' is an interface to the Controller that communicates with the View and the Model components of the architecture. It receives users actions from the View component and communicates with the Model to execute appropriate functions in response to the user's actions (for instance updating parameters).

2.4.5.3 View Interfaces

- 'View Screen' is an interface to the View that communicates with the Model component of the architecture in order to request updated data for display.

- 'User's Gestures' is an interface to the View that communicates with the Controller to notify the Controller of user's actions in order to respond to user's input.

2.5 Alternative Considered

Another alternative that was considered by the design team was to have the Controller component of the architecture call BLAST and deal with BLAST processing instead of the Model component. This was because the design team felt that the Model should strictly store data and not deal with processing, and it would be good to keep the BLAST processing class and database separate.

This alternative was not implemented because the team discovered that this implementation would make design and coding complicated. This implementation would cause data to be transferred repeatedly between the Controller and the Model components. This would cause difficulty for the maintenance team to maintain and extend the system in the future. Therefore, the team concluded to have the processing and data in the Model component instead. This way the client will largely only have to work with the Model component of the architecture to change anything non-GUI related.

THREE

Detailed Design

This section aims to describe each component of the MVC architecture. As each component of the architecture is complex, this section will explain each component separately. Under each component, there is a description of it, information on the data it is responsible for (including the type, visibility, and description of each datum), and also the important methods contained in it. English and pseudocode descriptions will be included with the description of each method.

The design team has focused more on the Model and the Controller components of the architecture as of now, because the business logic is of priority at this point of the design stage. The presentation logic will be discussed further in this document as the design grows.

3.0.1 Model Component

The main class in this component is 'Model Interface'. It acts as an interface between the Model and the rest of the architecture. A 'CSV Reader' module exists to read 'File's passed into the system. 'Reference File', 'Input File', 'Quality File', 'Grouped File', 'FASTA File', 'Tag File' and 'CSV File' are all modules that represent their corresponding file types. Each of these modules are subclasses of 'File'.

A 'Reference File' contains a 'Reference Line', which is simply a line from the file. Therefore, a 'Reference File' or 'Line' objects. Each 'Reference Line' object contains an identification (ID) and a shortread which represents a BLAST region.

Similarly, an 'Input File' contains an 'InputLine' and contains an 'Short Read'. A 'Short Read' contains an 'Identification', 'count' and an 'Input BLAST Region'. The 'Identification' represents the identification of that particular short read as specified by the user. The 'count' represents the number of times that 'Short Read' has occurred in the 'Input File'.

There is a 'PreProcessing Handler' that takes care of the conversion of data between objects for the pre-processing flow. 'Input Line' objects are passed into the module and used to generate 'Quality File', 'Grouped File' and 'Short Read' objects are passed into the module to generate a 'Tag File'.

A 'BLASTTool' module exists to take in the pathname to the BLAST alignment tool, pathname to the format tool and the BLAST parameters (represented as the module 'Parameters'), and generates 'FASTA File' objects so they can be passed into BLAST. This extends an AbstractAlignmentTool module.

A ‘BowtieTool’ module is created to allow future extension of the Bowtie alignment tool.

‘FileNameExtensionFilter’ is used to filter files with specific extension. This is required so that Java 1.5 on Mac can be supported.

‘Group’ is used to store the name entered by the user and a list of IDs.

3.0.2 Controller Component

The controller is responsible for attaching listeners to the appropriate View components and maps these actions to the Model component. The Controller constantly listens to the View component for user’s input.

Such listeners may include waiting for the right tab to be selected (InputTabChangeListener) or waiting for a file to be selected (InputFileListener).

‘AlignmentSelectListener’ listens to the user selection of the alignment tool they want to use.

‘BLASTListener’ calls BLAST and sends the ’FASTA File’ objects and ’Parameters’ in order to generate a ’Results’ object. A ’CSV Writer’ module converts standard BLAST output into a ’CSV File’ object.

There are numerous modules handles the navigation buttons for each individual Panel such as ‘GroupListener’, ‘PArametersListener’ and ‘RefBLASTRegionSelectionListener’

‘ModeSelectListener’ listens to the two buttons that set the flow of the program.

‘QualityListener’ listens to the quality values that the user has entered.

‘RefColumnHeaderChooserListener’ listens to the user selection of column headers that they want.

3.0.3 View Component

The View component is responsible for the creation and display of the GUI for the end user. The View component is made up of several sub-components which hold their relevant data which is often requested from the Model.

Firstly, there is a InputPanel class, which is a JPanel who holds various other JPanel’s which are related to the InputFiles. Such JPanels are FileSelection, RegionSelection, IDSelection and so on.

There is also a ReferencePanel class, which again holds JPanels relevant to ReferenceFiles. Such JPanels are FileSelection, RegionSelection, OutputColumns and etc.

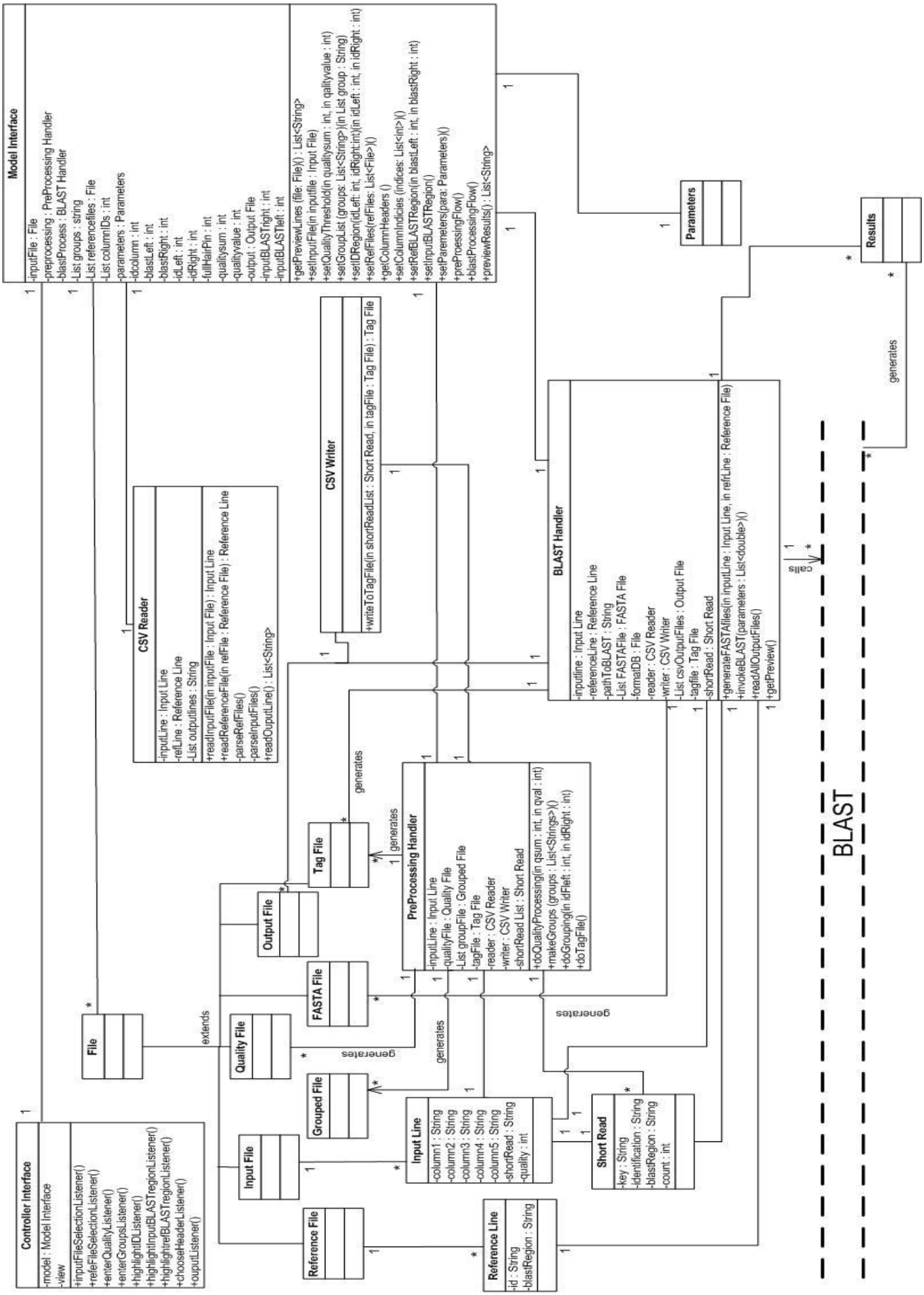
FileSelection JPanels allow the user to select a file of the relevant file, while RegionSelection and IDSelection Panels will allow the user to select appropriate regions to be stored in the Model component.

There is a BLASTPanel which allows the user to set and edit parameters before calling the alignment tool from the same JPanel.

Finally, an OutputPanel will be used to display the results from the execution to BLAST.

3.1 Class Diagram

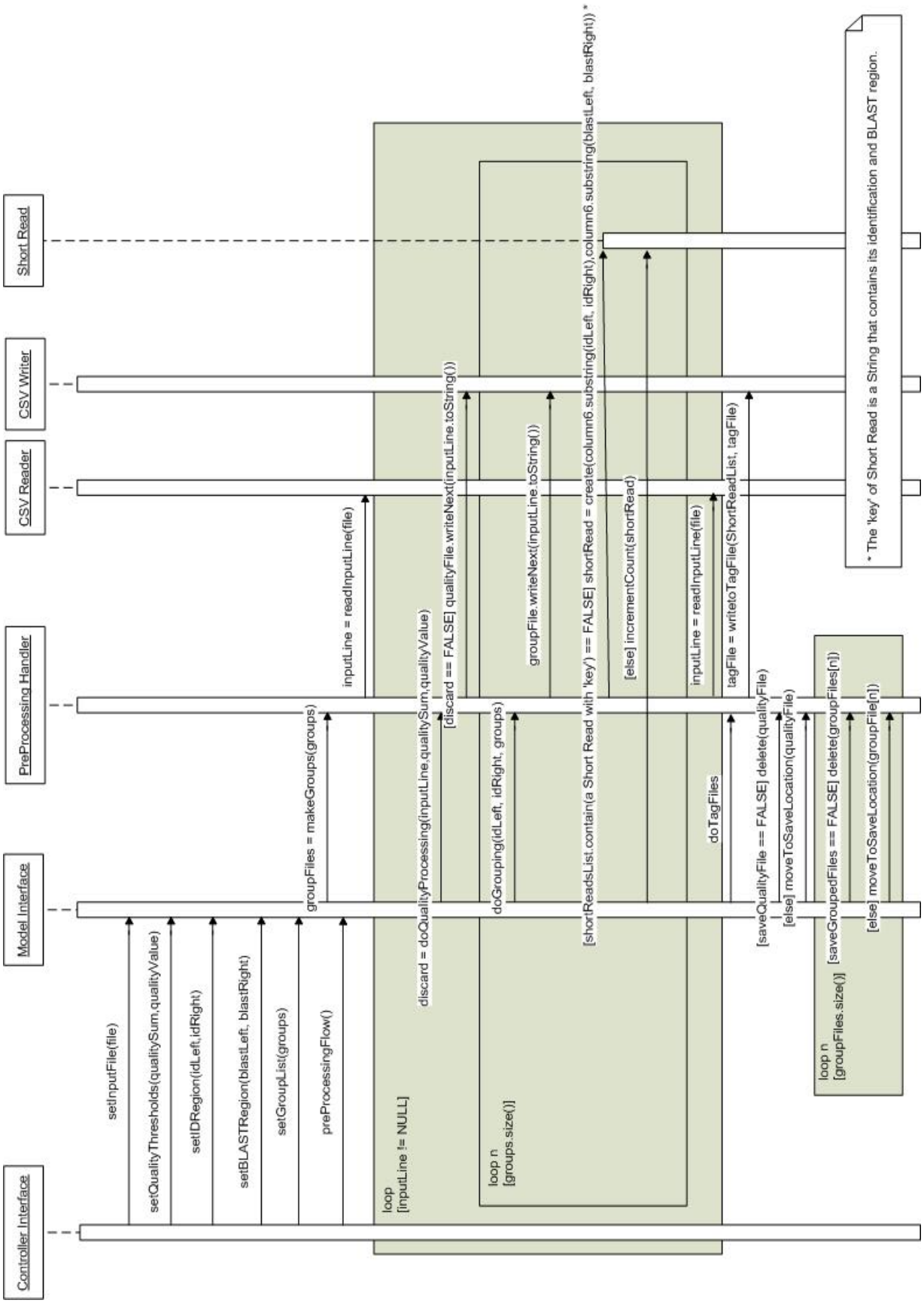
The class diagram represents the Model and the Controller components of the architecture. It shows classes that exist in each component and important attributes and operations carried out by each of the classes and the dependencies between them.


Model Interface
-inputFile : File
-preprocessing : PreProcessing Handler
-blastProcess : BLAST Handler
-List groups : string
-List referencefiles : File
-List columnIDs : int
-parameters : Parameters
-idcolumn : int
-blastLeft : int
-blastRight : int
-idLeft : int
-idRight : int
-fullHairPin : int
-qualitysum : int
-qualityvalue : int
-output : Output File
-inputBLASTright : int
-inputBLASTleft : int
+getPreviewLines (file: File)() : List<String>
+setInputFile(in inputfile : Input File)
+setQualityThreshold(in qualitysum : int, in qalityvalue : int)
+setGroupList (groups: List<String>)(in List group : String)
+setIDRegion(idLeft: int, idRight:int)(in idLeft : int, in idRight : int)
+setRefFiles(refFiles: List<File>)()
+getColumnHeaders ()
+setColumnIndicies (indices: List<int>)()
+setRefBLASTRegion(in blastLeft : int, in blastRight : int)
+setInputBLASTRegion()
+setParemeters(para: Parameters)()
+preProcessingFlow()
+blastProcessingFlow()
+previewResults() : List<String>
CSV Reader
-inputLine : Input Line
-refLine : Reference Line
-List outputlines : String
+readInputFile(in inputFile : Input File) : Input Line
+readReferenceFile(in refFile : Reference File) : Reference Line
-parseRefFiles()
-parseInputFiles()
+readOuputLine() : List<String>
CSV Writer
+writeToTagFile(in shortReadList : Short Read, in tagFile : Tag File) : Tag File
BLAST Handler
-inputline : Input Line
-referenceLine : Reference Line
-pathToBLAST : String
-List FASTAFile : FASTA File
-formatDB : File
-reader : CSV Reader
-writer : CSV Writer
-List csvOutputFiles : Output File
-tagfile : Tag File
-shortRead : Short Read
+generateFASTAfiles(in inputLine : Input Line, in refLine : Reference File)
+invokeBLAST(parameters : List<double>)()
+readAllOutputFiles()
+getPreview()
Parameters
Results
Controller Interface
-model : Model Interface
-view
+inputFileSelectionListener()
+refeFileSelectionListener()
+enterQualityListener()
+enterGroupsListener()
+highlightIDListener()
+highlightInputBLASTregionListener()
+highlightrefBLASTregionListener()
+chooseHeaderListener()
+ouputListener()
File
Input File
Output File
Reference File
Quality File
FASTA File
Grouped File
Tag File
PreProcessing Handler
-inputLine : Input Line
-qualityFile : Quality File
-List groupFile : Grouped File
-tagFile : Tag File
-reader : CSV Reader
-writer : CSV Writer
-shortRead List : Short Read
+doQualityProcessing(in qsum : int, in qval : int)
+makeGroups (groups : List<Strings>)()
+doGrouping(in idFleft : int, in idRight : int)
+doTagFile()
Input Line
-column1 : String
-column2 : String
-column3 : String
-column4 : String
-column5 : String
-shortRead : String
-quality : int
Short Read
-key : String
-identification : String
-blastRegion : String
-count : int
Reference Line
-id : String
-blastRegion : String
extends
generates
generates
generates
generates
generates
calls
generates
BLAST

3.2 Sequence Diagrams

The way classes in each component interact with each other is demonstrated in the form of sequence diagrams. The sequence diagrams focus more on the interaction between the Model and the Controller of the architecture. The Controller is assumed to be 'listening' to the user's interaction with the View at all times, as an MVC architecture is supposed to be like.

3.2.1 Preprocessing Flow



The Controller Interface class in the Controller sets the attributes - inputFile, quality-Sum, qualityValue, idLeft, idRight, blastLeft, blastRight, and groupList - in the Model

Interface of the Model by passing arguments into their corresponding setters.

The Model Interface calls the makeGroups() method in the PreProcessing Handler to create a list of Grouped Files - groupFiles - and initialising them to NULL.

The Controller Interface calls preProcessingFlow() in the Model Interface. This method contains a list of methods to carry out in sequence. First of all, the input file - file - is read and the first line of the file is set to the attribute inputLine in PreProcessing Handler. If inputLine is not NULL, i.e. the file is readable and it is not the end-of-file, the program enters a loop.

The Model Interface calls doQualityProcessing() in PreProcessing Handler. This method decides whether that line of input - inputLine - should be discarded or not based on the user-specified quality thresholds. If it is not to be discarded, it is appended to the end of a file - qualityFile - using the writeNext() method. This file will be created by writeNext() if it doesn't exist. If it is to be discarded, it will simply not be written into the file.

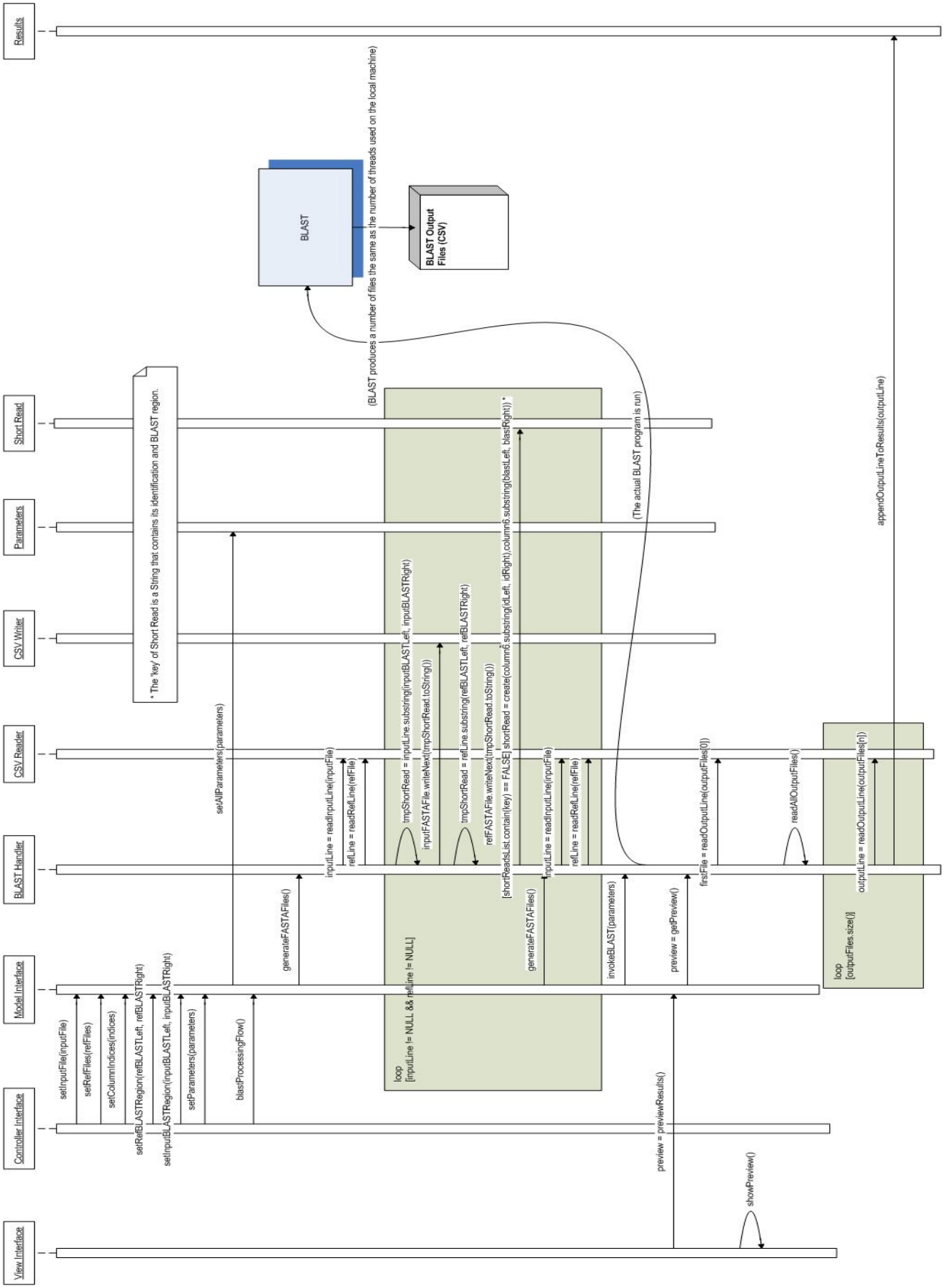
Identifications are to be appended to their respective Grouped Files now. doGrouping() checks the identification of inputLine and appends it to the end of the file that represents the group the identification belongs to, as specified by the user.

PreProcessing Handler contains a list of Short Read objects. Short Reads each contains an attribute - key - that is a String containing the identification and BLAST region of a particular inputLine. If there is no existing Short Read object with the same key as the key that would be generated from inputLine's identification and blastRegion, then a new Short Read object is created. Otherwise, the Short Read's count is incremented with incrementCount().

Once all the Grouped Files have been processed, a new line of input is read and the outer loop runs again. The outer loop ends once the end-of-file is reached.

Model Interface then calls the method doTagFiles() which is implemented in the PreProcessing Handler to write into the Tag File. The doTagFiles() method calls writeToTagFile() method implemented in the CSV Writer. It takes in the list of Short Read objects and a TagFile that is going to be written into. The method returns a Tag File that contains the ID and the count of each Short Read in the list.

3.2.2 BLAST Processing Flow



The Controller Interface begins by setting inputFile, refFiles, columnIndices, refBLASTRegion, and inputBLASTRegion in the Model Interface using the given arguments. Model Interface has a Parameters object that was created when the Model Interface was created. The attributes inside Parameters will be set by setAllParameters(), which is called when setParameters() is called.

The Controller Interface class calls blastProcessingFlow() in the Model Interface. This begins a list of method calls. Firstly, the Model Interface calls generateFastaFiles() in the BLAST Handler. This begins subsequent calls to CSV Reader. A line of the input file - inputFile - is read into inputLine, and a line of the reference files is also read into refLine.

While either of both files is readable and not end-of-file, a temporary variable is set to the BLAST region of the inputLine and then written into a FASTA File object called inputFASTAFile. Afterwards, this variable is given the BLAST region of refLine and then written into a FASTA File object called refFASTAFile. If any one of the files is end-of-file, then the corresponding writeNext() will simply not be called.

Short Read objects are created or have their count attributes incremented based on the key attribute as explained in the previous sequence diagram.

generateFASTAFiles() is called again and a new line of inputFile and refFiles are read into inputLine and refLine. The loop repeats until all input and reference files have been converted into FASTA.

Once the FASTA File objects have been completed, BLAST is called with a Parameters object. BLAST generates output files in CSV format. The number of output files is the number of groups specified by the user.

The first output file will be read and passed to the View Interface to be previewed. Afterwards, all the lines of output files are read and added to a list of Strings called results.